Computer Programming

Course 66111

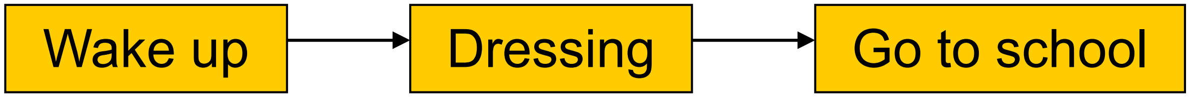
Instructor: Eng. Muhannad Al-Jabi

What do we mean by program?

- It is a set of ordered instructions to do a certain task.

Instruction A is executed before instruction B, as long as instruction A is located before instruction B.

- For example, some instructions of your daily program:-



Stages of program development

- Problem statement.
- Program design.
- Program writing.
- Program compilation.
- Program linking.
- Program loading.
- Program execution and testing.

Problem Statement

- What is the problem? Why the program is needed?

- What is the scope and limitations (in time, and accuracy) that can be used to solve the problem?

Program Design

- Find a suitable algorithm to solve the problem.
- Draw the flowchart of that algorithm.

What does it mean?

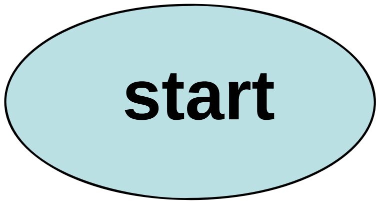
Flowchart

- Flowchart is the step-by-step solution of a problem, using suitably annotated geometric figures connected by flow lines for the purpose of designing.
- Such that, program instructions are categorized into different categories, and each category has different geometric figure.
- Each instruction is presented by the geometric figure of its category.

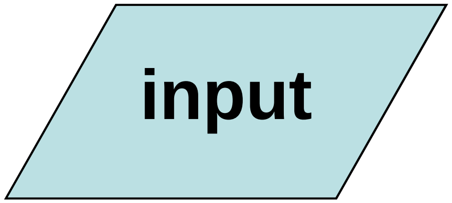
Flowchart

Instruction category Geometric figure

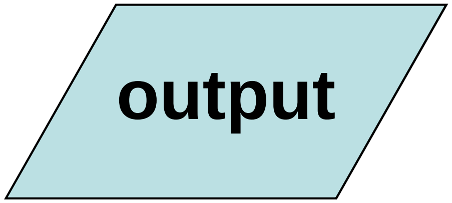
- Starting program



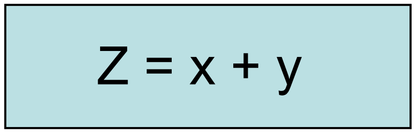
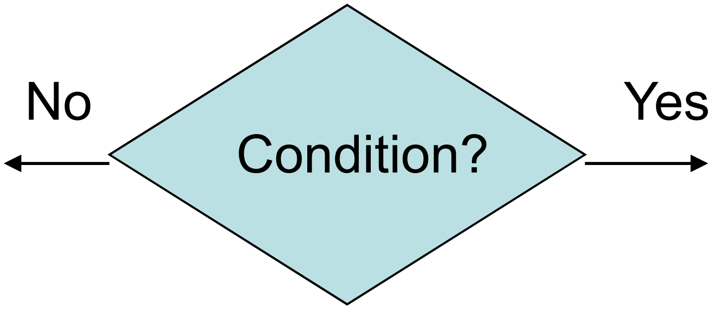
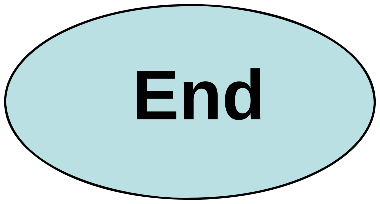
- Input



- Output



Flowchart

Instruction category	Geometric figure
Process	 Z = x + y
Conditional instruction	 Condition? No Yes
End program	 **End**

Flowchart example 1

Draw the flowchart for a program that reads two numbers and prints the sum of these two numbers.

Flowchart example 1

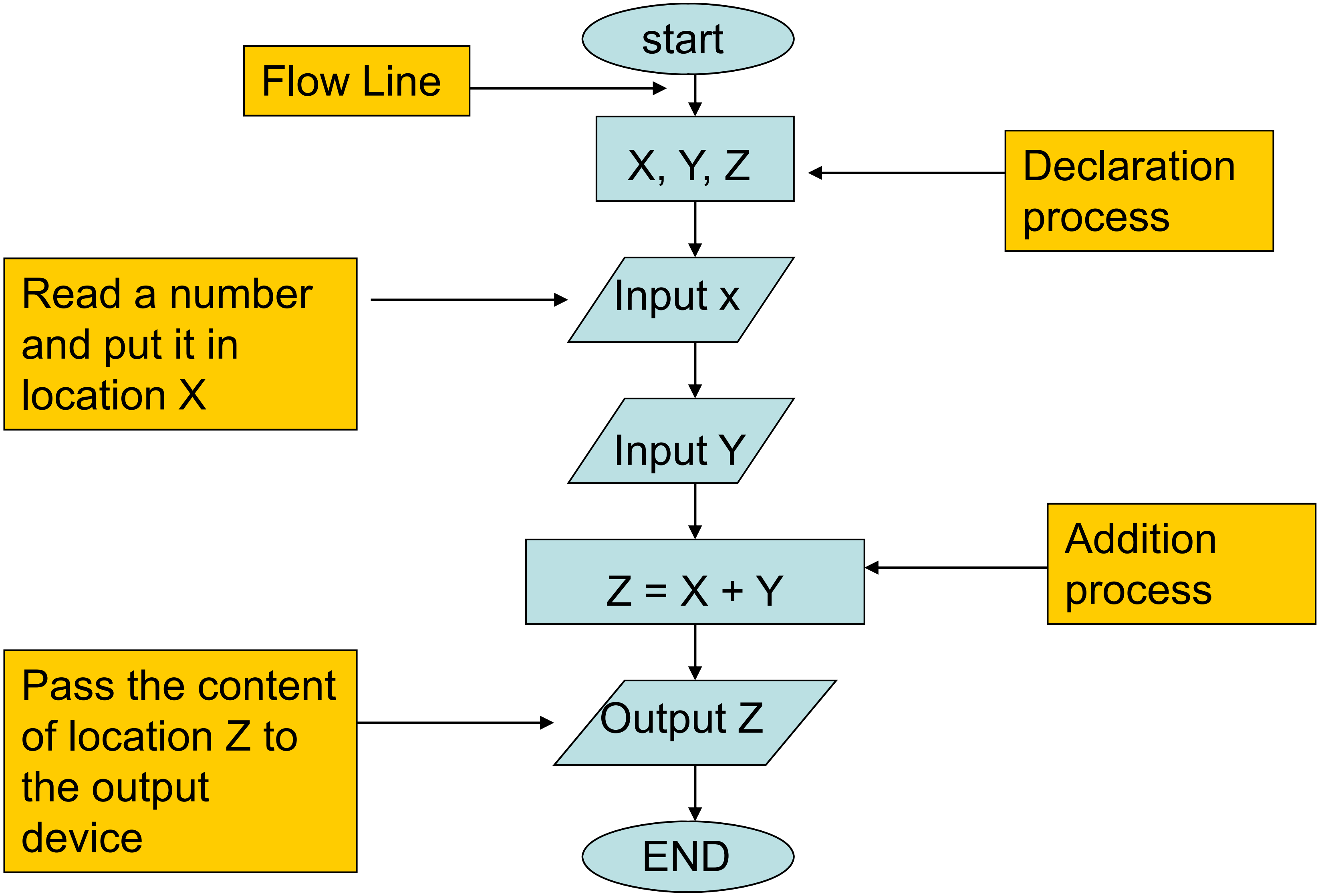
- The program must read two numbers.
- But where can the program store these numbers?
- The answer is, in two memory locations have been previously allocated by a process called Declaration.

locate a location in memory and assign it a name for later use

Flowchart example 1

- The sum can be calculated and stored in another location that has been declared.
- Now the program can print the result by passing it to the output device.

Flowchart example 1



start

Flow Line

X, Y, Z

Declaration process

Read a number and put it in location X

Input x

Input Y

Z = X + Y

Addition process

Pass the content of location Z to the output device

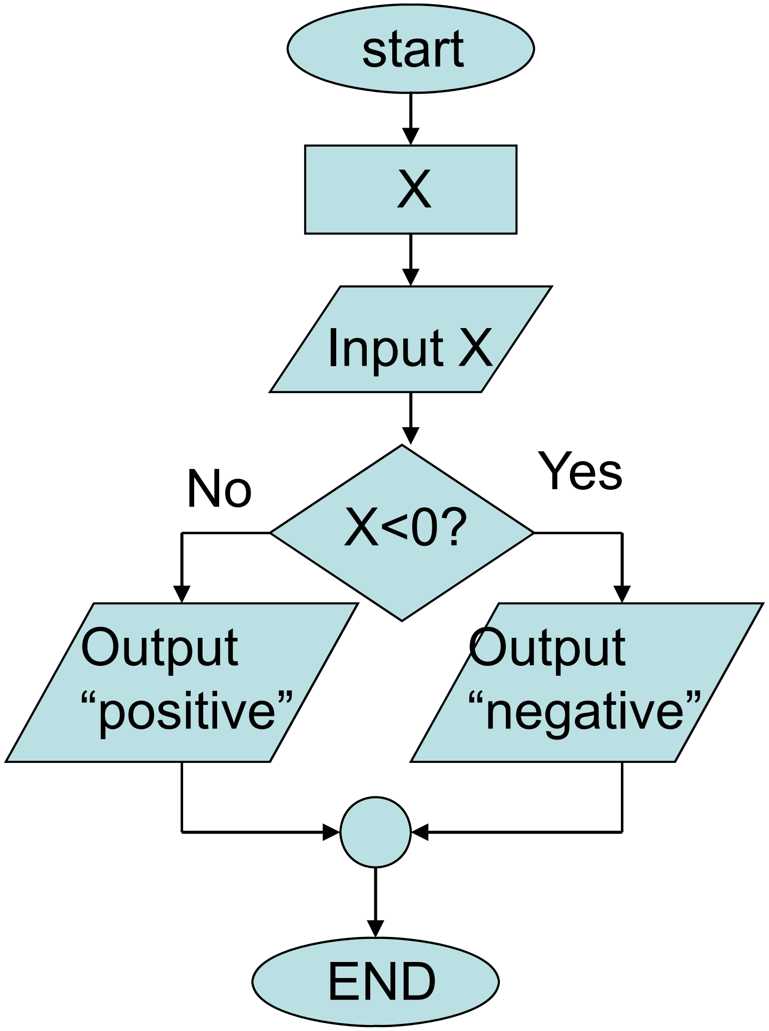
Output Z

END

Flowchart example 2

Draw the flowchart for a program that reads a number, and determines whether the number is positive or negative.

Flowchart example 2



Why did we put the quotations around the words positive and negative?

When we need to print a word or a sentence as it is, we must put it between quotations.

Flowchart example 3

Draw the flowchart for a program that evaluates the average for N input numbers

Flowchart example 3

- How many memory locations do we need?
- Do we need N memory locations?
- Take in consideration that N is an input number decided by the user. So, the value of N is user dependent.

Flowchart example 3

- From the equation Avg = Sum / N we can decide that we need 3 locations to start with:-
- Avg
- Sum
- N

Flowchart example 3

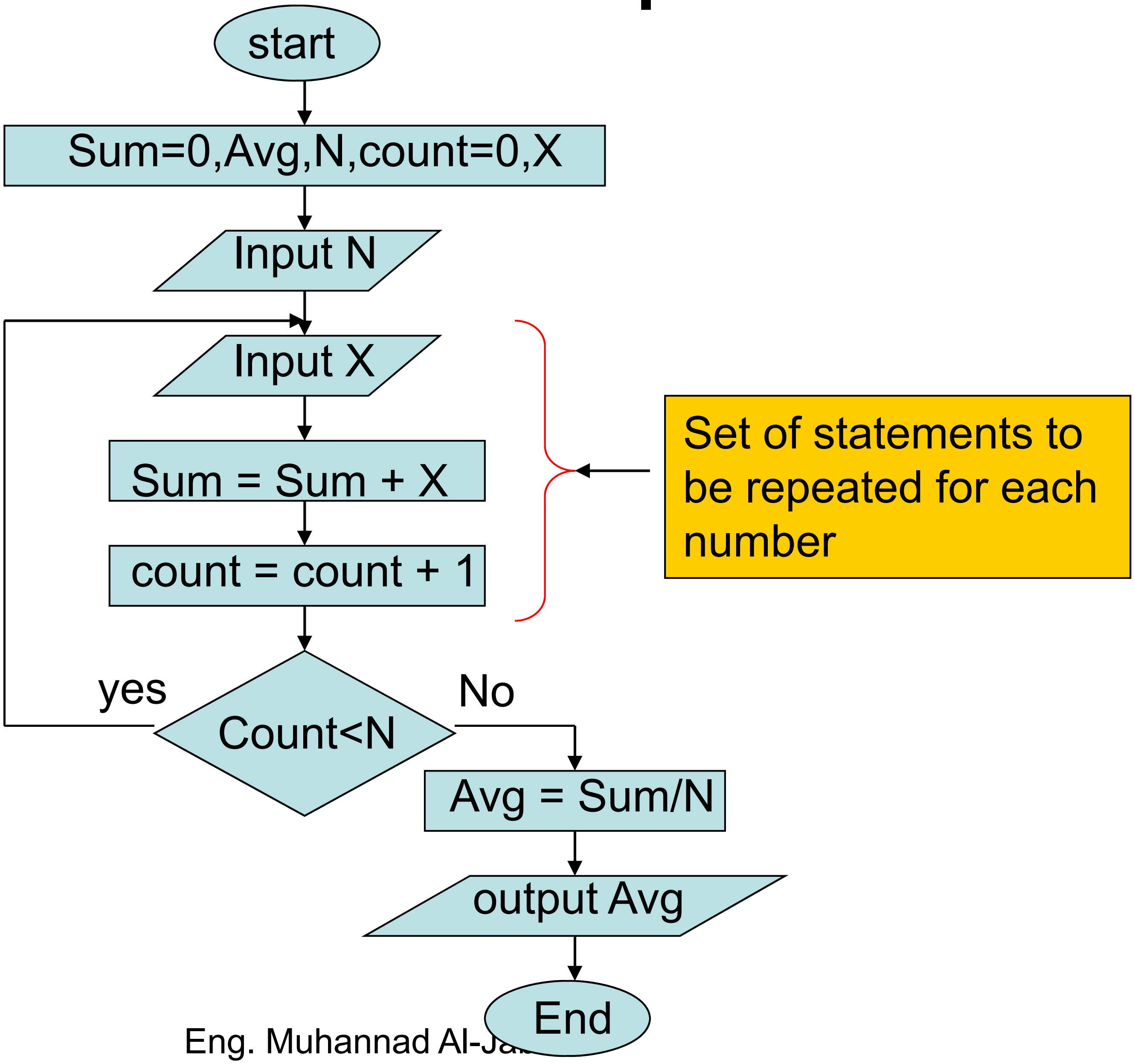
Since we can’t allocate N locations, we need a location x to use while reading the N numbers. And repeat the following statements for each number:-

- Read the number and put it in location x.
- Add the content of x to the content of sum.

Flowchart example 3

- How do we know that we have repeated the previous two statements for N times?
- We need a counter that already initialized to zero, and increment the counter for each number.

Flowchart example 3



start

Sum=0,Avg,N,count=0,X

Input N

Input X

Sum = Sum + X

count = count + 1

Count<N

yes

No

Avg = Sum/N

output Avg

End

Set of statements to be repeated for each number

Program writing

- We will use Microsoft Visual Studio 6.0.

It can be used for:-

- Program writing.
- Program compilation.

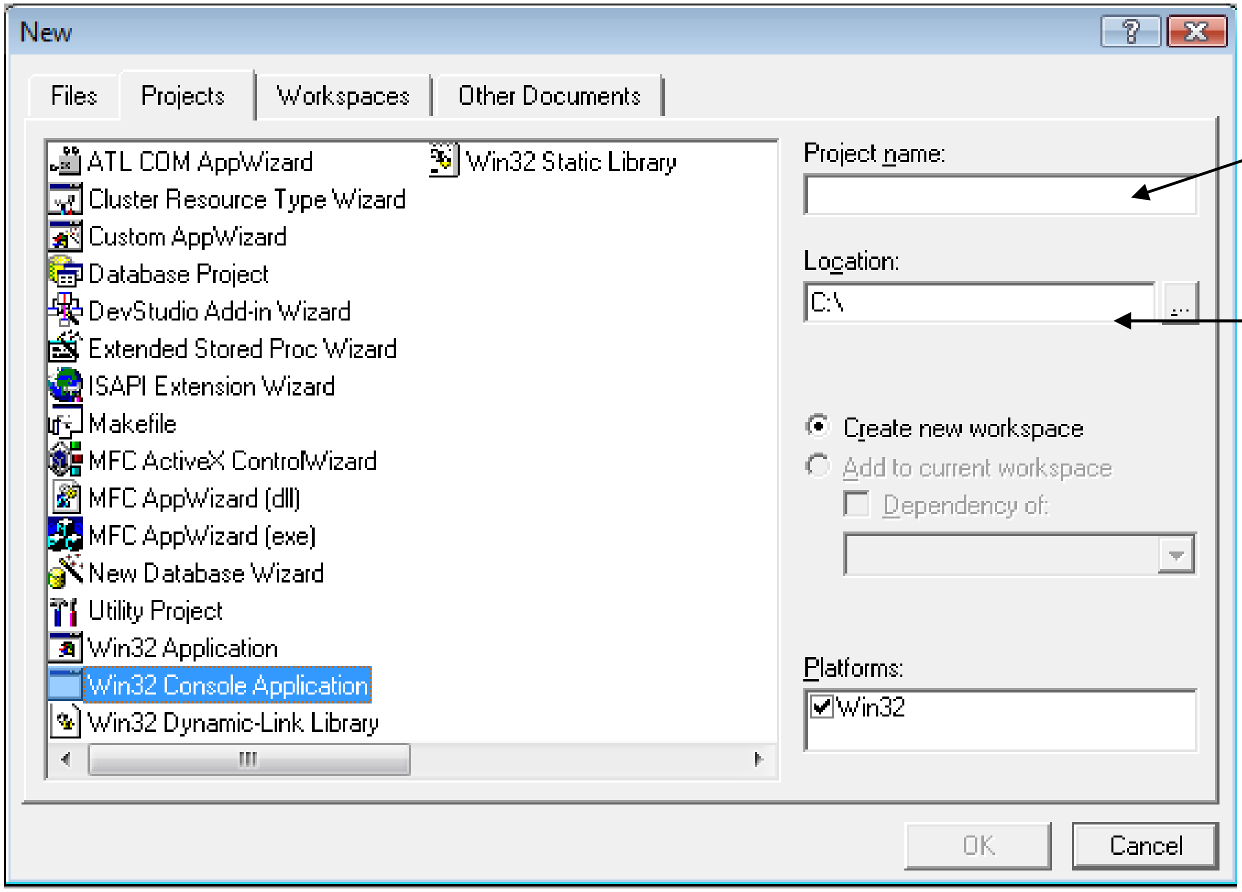
How to start writing a program?

- Go to all programs menu from desktop.
- Select Microsoft Visual Studio 6.0.
- Select Microsoft Visual C++ 6.0

How to start writing a program?

- From file menu select new.
- Select the projects tap.
- Select the win32 console application

How to start writing a program?



Project name:

Location: C:\

● Create new workspace

○ Add to current workspace

□ Dependency of:

Platforms: ☑ Win32

The project name

Where do you want to store your project?

After this point always press the ok bottom

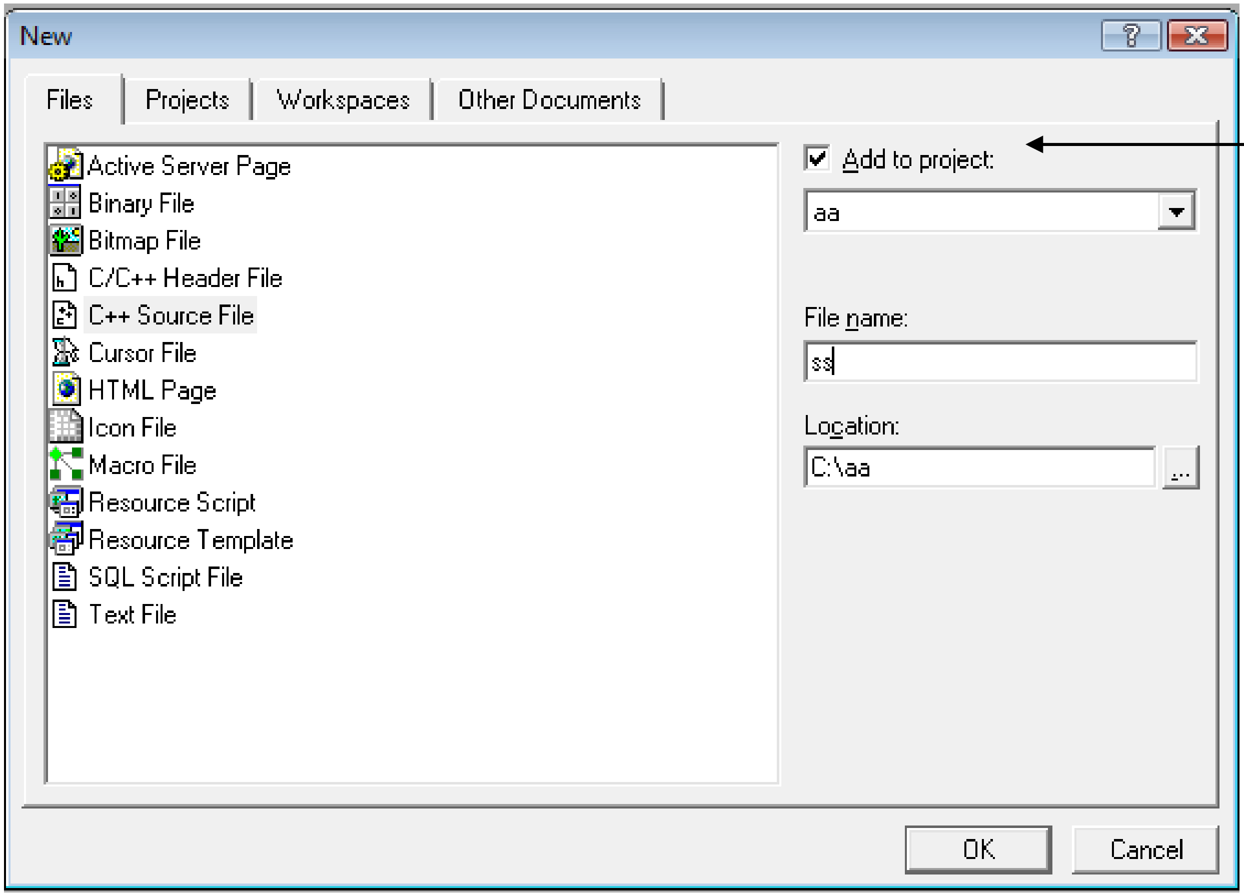
How to start writing a program?

- From the file menu select new.

- Select the file tap.

- Select the c++ source file option.

How to start writing a program?



☑ Add to project:
aa

File name:
ss

Location:
C:\aa

Don’t forget to tick this, in order to specify to which project does your file belong.

Any project may contain many files. But only one can contain the main function.

The extension of any C++ source file is cpp

Example #1

write a C++ program that prints a hello word on the screen.

Example #1

```
#include<iostream.h>
void main()
{
cout<<“Hello”;
}
```

The main function is part of every C++ program. And is called the main block.

This file must be included for any program that outputs data to the screen or inputs data from the keyboard.

C++ programs begin executing at the first statement after the left brace of the main and finish executing at the right brace of the main if there is no return statement.

Example #1

```
#include<iostream.h>
int main()
{
cout<<“Hello”;
return 0;
}
```

The main function is either of type int and contains the return statement or of type void and has no return statement. return 0 indicates that the program ended successfully.

There is must be a semicolon (;) at the end of each C++ statement

Escape sequence

- In some cases you want to break a sentence into many lines on the screen.

- For example, you want to print the “hello” word something like this: he
  llo

  Which means that the hello word has been broken into two lines.

Escape sequence

C++ statement contains escape sequence	output
cout<<“hello\n word”;	hello word
cout<<“hello\t word”;	Hello word
cout<<“hello\r word”;	word
cout<<“hello\a”;	hello You will hear a beep
cout<<“hello\\word”;	hello\word
cout<<“hello\” word”;	hello “ word
cout<<“hello”<<endl<<“word”;	hello word

Single line and multi-lines comments

- If we want that the compiler to ignore a certain line, we must put // before that line.

- If we want that the compiler to ignore a group of lines, we must put /* before the beginning of the first line. And we must put */ after the end of the last line.

- Commenting a program is useful for program readability

Variable types in c++

Variables: locations in memory where values can be stored.

The function **sizeof(*data type*)** can be used to see the size

Data type	description	Size in bytes
int	Integer	4
float	Single floating point number	4
double	Double floating point number	8
char	Character	1
bool	Logical; either true of false	1

Variable naming

Variable name must start with

- Letter

or

- Underscore

or

- $

Variable name cannot start with a digit

But, the rest of the variable name may contain letters, underscores, dollar signs or digits

Variable declaration

- You can declare the variable anywhere you want in your program.
- You must declare the variable before you can use it.
- You can declare each variable in one declaration statement.
- Also you can declare multiple variables of same type in one declaration statement.

Declaration statement structure

- Single variable declaration statement.

  data_type variable_name;

  Example : int x;

- Multiple variables declaration statement

  data_type var1,var2,var3,…….varn;

  Example : int x,y,z;

- You can give the initial value to the variable (initialize the variable) while declaring it

  Example: int x=6;

Example #2

Write a c++ program that reads two integer numbers and prints the result.

- The program must tell the user when to enter each number by print a message likes “enter the first number now please”.
- The program must print the result in this manner : the result =   12233……

Example #2

```
#include <iostream.h>
void main(){
int num1,num2, result;
cout<<“enter the first number plz”<<endl;
cin>> num1;
cout<<“enter the second number plz”<<endl;
cin>>num2
result = num1+num2;
cout<<“the result  =\t”<<result<<endl;
}
```

declaration

Input the first number and put it in location num1

Pass the content of location result to the output device

Example #3

What is wrong in this code (part of a program)?

```
int x=15;
X=10;
Cout<<x<<endl;
```

X is not declared, X is not the same as x

Cout is not the same as cout

So, C++ is case sensitive language

Arithmetic

- *   for multiplication
- /   for division
- -   for subtraction
- +   for addition
- % modulus operator returns remainder

Arithmetic

Rules of operator precedence

- Operators in parentheses evaluated first
- Exponential operators applied next

  example: x^y
- Multiplication, division, modulus applied next from left to right
- Addition and subtraction applied last from left to right

Integer division and float division

- int / int :the result is integer
- float /float : the result is float
- int/float: the result is float
- float/int: the result is float

Example: the result of 6/4 is 1

but the result of 6.0/4 is 1.5

Casting

- Treat a variable of type1 as if it is of type2.

(**type2**)*varibale of type1*

- It is mostly used when we don’t need an integer division to truncate the remainder.

Example:
```
int x=5,y=6;
float z;
z=(float)y/x;
```

Treat y as float so the division is float division and z=1.2.

If we remove the casting, the division becomes integer division and z=1;

ASCII values

- Each character can be treated as integer or as character.
- Each character is a one byte integer.
- Each character has its ASCII value.

  ‘a’…’z’ -> 97…122

  ‘A’…’Z’ -> 65…90

Example: cout<<(char)65; will print A

cout<<(int)’97’; will print a

Example #4

Write a C++ program that reads a small letter character and prints the upper case of that character.

Example #4

```
#include<iostream.h>
void main(){
char x;
cin>>x;
cout<<(char)(x-32)<<endl;
}
```

32 is the difference between the uppercase and the smaller case ranges

When a character exists in a mathematical equation, its ASCII value is automatically substituted

Assignment operators

- Addition assignment operator

  c = c + 3 can be written as c += 3

- Other operators can be written in the same manner

c = c *3	c*=3
c = c – 3	c-=3
c = c / 3	c/=3
c = c % 3	c%=3

Increment and decrement operators

- Increment operator is used to increment the variable by one in the form like

  x++ or ++x

This is equivalent to x+=1 and x=x+1

- Decrement operator is used to decrement the variable by one in the form like

  x--   or --x

Preincrement and postincrement

- Preincrement: variable changed before used in expression. ++x

- Postincrement: variable changed after used in expression. x++

Preincrement and postincrement

- What is the output of the following code?

```
int x = 3;
cout<<x++<<endl;
cout<<++x<<endl;
```

3, because x is incremented after the first print operation

5 ,because after the first print operation x is 4

And now x is incremented before the second print operation to become 5 and it is printed as 5

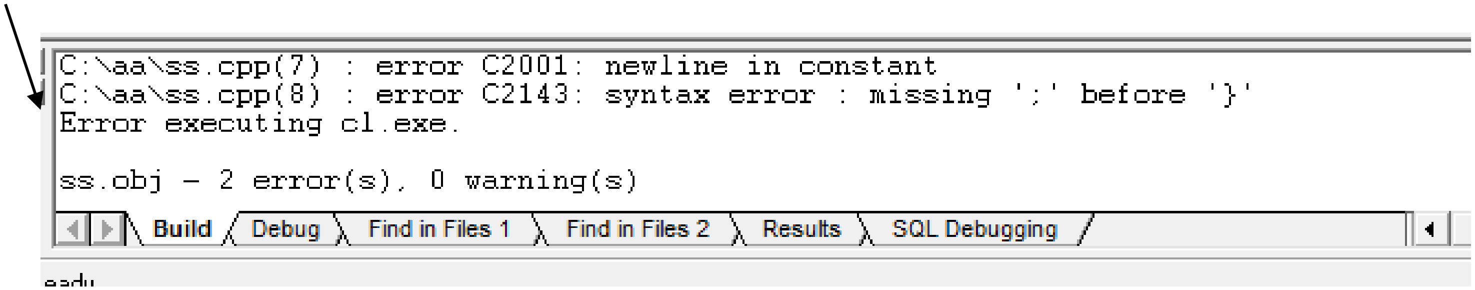
Compile a c++ program

- To compile a c++ program, press ctrl+F7
- If there is no errors, press ctrl+F5 to run the program.

How to find compilation errors

- In the window bellow the page of program editing.
- Go to the error name using right slide bare
- When you double click on the error, a notification arrow will point to the line that contains the error.
- Solve the first error first then the second and so on.

The window looks like this if there are some errors



```
C:\aa\ss.cpp(7) : error C2001: newline in constant
C:\aa\ss.cpp(8) : error C2143: syntax error : missing ';' before '}'
Error executing cl.exe.

ss.obj - 2 error(s), 0 warning(s)
```

Formatted output statement printf()

- The header file stdio.h must be included.

- printf statement consists of two parts:

1. The format part

2. The variables list part

Formatted output statement printf()

- Example:

Format part

Variables list part

```
int x=5,y=7;
printf(“x=%d\t y=%d\n”,x,y);
```

%d indicates that an integer number from the variables list will be printed in this location.

%f for float   %c for character

Formatted output statement printf()

- Example:

```
int x=5,y=7;
printf(“x=%10d y=%10d\n”,x,y);
float z=5.5;
printf(“z=%5.3f\n”,z);
```

Minimum number of spaces to print the integer number. If the number is shorter than the specified number of digits, it is padded with blank spaces

5 for width and 3 for precision

z will be printed as 5.500

Formatted input statement scanf()

Example:

```
 int x,y;
scanf(“%d%d”,&x,&y);
```

In scanf statement, any variable must be preceded by &

Scope of the variable

- The area where a variable is declared and can be accessed is referred to as the scope of the variable.
- You can create and define your own scope.
- Each scope starts with left bracket { and end with right bracket }, this scope is called block.

example

what is wrong in the following program?

```
#include<iostream.h>
void main(){
int x=5;
     {int y=9;
      cout<<x<<endl;
     }
 cout<<y<<endl;
}
```

Outer scope

inner scope

Any variable declared inside the outer scope can be accessed from the inner scope. But that declared inside the inner cannot be accessed from the outer.

y is defined inside the inner scope not inside the outer scope

Equality and relational operators

Relational operators	C++ condition
x is less than y	x<y
x is less than or equal y	x<=y
x is greater than y	x>y
x is greater than or equal y	x>=y
Equality operators	
x is equal to y	x==y
x is not equal to y	x!=y

If selection structure

```
 if(condition){
Statement or group of statements;
}
```

only if the condition is true, the enclosed statement or group of statements will be executed.

If-else selection structure

```
if( condition ){
First group of statements;
}
else{
Second group of statements;
}
```

The first group will be executed when the condition is true, and the second group will be executed when the condition is false.

example

write a c++ program in order to test wither an input number is positive or negative.

example

```
#include<iostream.h>
void main(){
int x;
cin>>x;
if(x<0){//if
        cout<<“negative”;}//if
else{//else
cout<<“positive”;}//else
}
```

These comments are preferred for brackets tracking. In order to avoid forgetting to close any block.

if-else-if structure

```
if(condition1){
First group of statements;
}
else if(condition2){
Second group of statement;
}
```

Second group is executed if and only if condition1 is false and condition2 is true

Example

What is the output of the following code?

```
int x=5,y=7;
If(x>0)cout<<“x is greater than zero\n”;

else if(y==7)cout<<“y=7\n”;
else cout<<“bye\n”;
```

Brackets are optional for blocks that contain only one statement

Else statement always belongs to the last if statement. And the block after else statement is executed only when the last if statement is tested and its condition is false.

Output is:

```
x is greater than zero
```

Example

What is wrong in the following code?

```
int x=10;
if(x==10)cout<<“x=10\n”;
cout<<“hi\n”;

else cout<<“x is not equal to 10\n”;
```

You can't put any statement between the if block and the else statement.

Conditional operator

- Three arguments: condition, value if true, value if false.

Example1:

```
cout<<(grade>=60?”pass”:”fail”);
```

Example2:

```
x=(z>0?z:y);
```

Only one statement can be exist in each argument

Example3:

```
(x<5?cout<<“hello\n”:cout<<“hi\n”);
```

Logical operations

- && Logical AND

True if both conditions are true

```
if((x<60)&&(x<35))cout<<“35”<<endl;
```

- || logical OR

True if either conditions is true

```
if((x>100)||(x<0))cout<<“x is not a grade\n”;
```

Logical operations

- ! Logical NOT, logical negation.

  Returns true when its condition is false and vice versa.

```
if(x!=y)cout<<“x,y are not equal\n”;
```

This statement is equivalent to

```
if(!(x==y))cout<<“x,y are not equal\n”;
```

Logical operations

- && operations evaluated first

  Example: what is the output of the following code?

```
bool x=true, y=false;
cout<<(x||x&&y)<<endl;
cout<<(x&&y||0)<<endl;
```

1

0

&& operations evaluated before || operations.

Any number exists in any logical expression is treated as true, except zero is treated as false.

Example

- What is the output of the following code?

```
int x=5;
if(x=6)cout<<++x<<endl;
if(x++==7)cout<<x<<endl;
```

This condition is always true because it is an assignment operation

7

Test then increment

8

Switch multiple-selection structure

- Test integer and characters variables for multiple values.
- Series of case labels and optional default case.

Example:

```
switch (variable){
case value1:
Statements;
break;
case value1:
Statements;
break;
default:
statements;
}
```

Break is necessary to exit the switch block after the execution of any group of statements.

If there is no break statement, every statement after the value of the variable will be executed till a break statement is encountered or till the end of the switch block;

Example

Using the switch structure, write a c++ program that reads a grade as character; A,B,C,D,E

And prints:

“excellent” for A

“very good” for B

“good” for C

“fair” for D

“fail” for other characters.

Example

```
#include<iostream.h>
void main(){
char grade;
cin>>grade;
switch(grade){//switch
case ‘A’:cout<<“excellent\n”;break;
case ‘B’:cout<<“very good\n”;break;
case ‘C’:cout<<“good\n”;break;
case ‘D’:cout<<“fail\n”;break;
default: cout<<“fail\n”;
} //switch
}
```

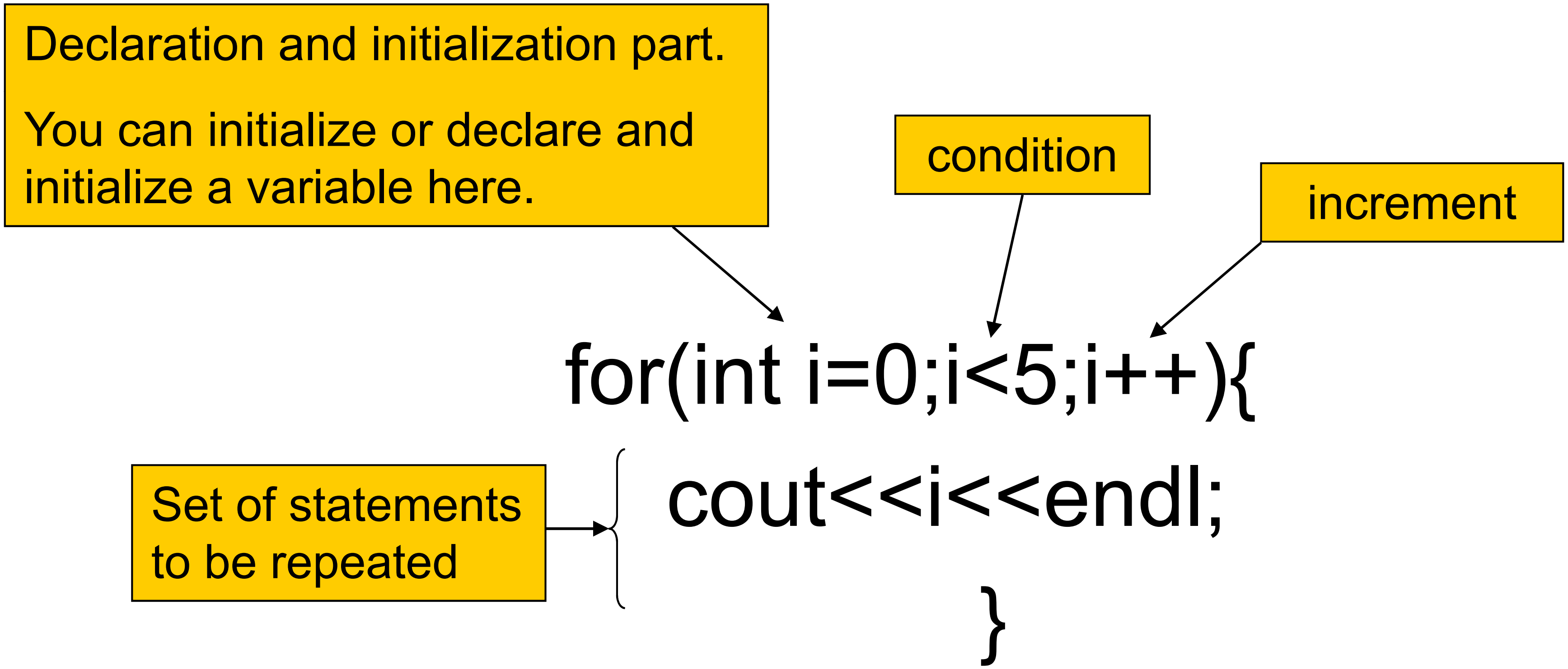
Loops

- Iterative loops: counter, initial value, condition, increment.

  The number of repetitions is predictable.

- Conditional loops: the condition is the only thing that controls the number of repetitions

for loop structure



Declaration and initialization part.

You can initialize or declare and initialize a variable here.

condition

increment

```
for(int i=0;i<5;i++){
    cout<<i<<endl;
}
```

Set of statements to be repeated

- Any part of the three parts can be empty.
- The condition part can be any condition.
- You can increment anything in the increment part

for loop structure

The same structure can be written like this:

Declaration or initialization is executed only for one time

In the for structure written in the previous slide, the condition is tested at the beginning of the repetition and the counter is incremented at the end of the repetition.

```
 int i=0;
for(;i<5;){
cout<<i<<endl;
   i++;
     }
```

Example

Write a c++ program that reads N input grades, and determines the average.

Hint: you can see the flowchart for this example written in the program design part.

Example

```
#include<iostream.h>
void main(){
float avg,x, sum=0;
int n;
cin>>n;
for(int i=1;i<=n;i++){//for
cin>>x;
sum+=x;
}//for
avg=sum/n;
cout<<“average=\t”<<avg<<endl;
}
```

Example

What is the output of the following code?

```
for(int i=0;i++<10;i++){
 cout<<i<<“\t”;
}
```

```
1	3	5	7		9
```

Because in each repetition, the counter is incremented two times:

1 Just after the condition has been tested.

2 At the end of the repetition

Example

What is the output of the following code?

```
for(int i=1;i<=3;i++){//for
switch(i){//switch
case 1:cout<<“hi\t”;
case 2:cout<<“hello\t”;
case 3:cout<<“bye\t”;
                    }//switch
cout<<endl;
}//for
```

```
hi          hello       bye
hello       bye
bye
```

It seems like that

“bye” is printed when the value of i is 1 or 2 or 3.

“hello” is printed when the value of i is 1 or 2

“hi” is printed only when the value of i is 1

Nested for structure

When there are two or more for structures inside each others, this called nested for structure.

The word nested can be used for any structure, when there are many blocks inside each others. Ex: for, if, while,…etc.

Example

Write a c++ program in order to print the numbers 1..100 ten numbers per line.

Such that, the first line contains the numbers 1..10, the second contains 11..20 and so on.

Example

```
#include<iostream.h>
void main(){//main
for(int i=0;i<10;i++){//outer for
   for(int j=1;j<=10;j++){//inner for
        cout<<10*i+j<<“\t”;
        }//inner for
cout<<endl;
}//outer for
} //main
```

For each repetition of the outer loop, the inner loop is repeated 10 times

So, the outer loop is for line transition, and the inner loop is for printing the numbers.

While repetition structure

- Action repeated while some condition remains true.

```
        while(condition){
       set of statements}
```

Example: what is the output of the following code?

condition

```
int x=2;
while(x<10){
x+=2;
cout<<x<<endl;
}
```

The output is

```
4
6
8
10
```

While repetition structure

- If the statements are repeated n times, the condition is tested n+1 times. Why?

- Because in the last time the condition will be tested, the condition will be false. So, no statement will be executed.

do/while repetition structure

- Similar to while structure.

  But it tests the condition at the end of each repetition, not at the beginning.
- It is possible for the loop body to be executed at least for one time.

Example: what is the output of the following code?

```
int x=5;
do{
cout<<“hello\t”<<x<<endl;
x++;
}while(x<5);
```

The output is:

```
hello	5
```

If the statements inside the loop are executed n times, then the condition is tested for n times

Example

Write a C++ program that evaluates the average for unknown number of input numbers. such that, as long as the input number is positive, the program takes another number. But if the input number is negative, the program will stop and it will exit.

how can you use the while structure?

How can you use the do-while structure?

Solution using the while structure

```
#include<iostream.h>
void main(){
float avg,sum=0,count=0,x=0;
while(x>=0){//while
cin>>x;
sum+=x;
count++;
} //while
avg= sum/count;
cout<<“the result=\t”<<avg<<endl;
}
```

For this example, we must initialize x by a temporary value that makes the while condition to be true.

Otherwise the statements inside the loop can not be executed.

Solution using the do-while structure

```
#include<iostream.h>
void main(){
float avg,sum=0,count=0,x;
do{
cin>>x;
sum+=x;
count++;
} while(x>=0);
avg= sum/count;
cout<<“the result=\t”<<avg<<endl;
}
```

Whatever the value of x, the statements inside the loop will be executed at least for one time.

Break statement

- Break statement breaks the loop to the outer block.

Example: what is the output of the following code?

```
for(int i=0;i<5;i++){
cout<<i<<endl;
if(i==3)break;
}
```

The output is:

```
0
1
2
```

Continue statement

- Continue statement ignores the rest of statements inside the current loop block, and continue the execution from the beginning of the next repetition.
- Example: write a c++ code to print the numbers from 1 to 10 except 4 and 6.

```
for(int i=1;i<=10;i++){
if((i==4)||(i==6))continue;
cout<<i<<endl;
}
```

Constant variables

- Its value can not be changed.
- Constant variables must be initialized.
- Example: const int x=10;
- Not valid statement: cont int y;

y must be initialized because it is constant variable

C++ reserved words

These words can not be used to name any variable in c++

C++ Keywords

Keywords common to the C and C++ programming languages

auto	**break**	**case**	**char**	**const**
continue	**default**	**do**	**double**	**else**
enum	**extern**	**float**	**for**	**goto**
if	**int**	**long**	**register**	**return**
short	**signed**	**sizeof**	**static**	**struct**
switch	**typedef**	**union**	**unsigned**	**void**
volatile	**while**			

C++ only keywords

asm	**bool**	**catch**	**class**	**const_cast**
delete	**dynamic_cast**	**explicit**	**false**	**friend**
inline	**mutable**	**namespace**	**new**	**operator**
private	**protected**	**public**	**reinterpret_cast**	
static_cast	**template**	**this**	**throw**	**true**
try	**typeid**	**typename**	**using**	**virtual**
wchar_t				

Example

What is the output of the following program?

```
#include<iostream.h>
int t(int a){return 2*a;cout<<“hello”<<endl;}
void main(){
cout<<t(3)<<endl;
}
```

The output is :6

We can notice that no statement after the return statement will be executed.

Arrays

- Consecutive group of memory locations.
- Same name and type.

To refer to an element in an array we must specify the array name and the position number of that element.

Example: ***name***[*position number*]

The positions of N-locations array are numbered from 0 to N-1

Arrays declaration

When declaring arrays we must specify:-

- Array type.
- Array name.
- Array size.

Type ***name***[*array size*];

We can declare multible arrays of the same type by:

type name1[size1], name2[size2],…;

Arrays declaration

Also we can specify the size of the array as follows:-

Type name[]={element1,element2,….};

By this way the size of the array is specified by the number of elements in the list. We can see that we fill the array at declaration.

Example

From the following array declaration, what will be the content of the array after the program has been compiled and executed?

```
int x[5]={1};
```

The content of x is 1,0,0,0,0 that means the remaining locations are filled by zeros.

Example

From the following array declaration, what will be the content of the array after the program has been compiled and executed?

```
int x[5]={1,2,3};
```

The content of x is 1,2,3,0,0.

Arrays manipulation

Arrays can be filled at declaration or at execution. Such that:-

```
int x[5];
cin>>x[0]>>x[1]>>x[2]>>x[3]>>x[4];
```

Or by using loop:-

```
for(int i=0;i<5;i++)cin>>x[i];
```

Arrays manipulation

Also arrays are printed element by element either by printing each element explicitly, or by using loops.

Example:

```
cout<<x[0]<<x[1]<<x[2]…..<<x[N-1];
for(int j=0;j<N;j++)cout<<x[j];
```

Elements are implicitly mentioned through changing the index j

All elements to be printed are explicitly mentioned

Arrays manipulation

- Arrays can be printed in order: element i is printed before element i+1.

Example: for(int i=0;i<N;i++)cout<<x[i];

- Arrays can be printed in reverse order: element N is printed first then N-1 and so on.

  for(int i=N-1;i>=0;i--)cout<<x[i];

Note that: element N is located in location N-1

Arrays manipulation

- Also arrays can be printed in reverse order by:

```
for(int i=0;i<N;i++)cout<<x[(N-1)-i];
```

- How can we print the elements in even locations?

```
for(int i=0;i<N;i+=2)cout<<x[i];
```

- How can we print the elements in odd locations?

```
for(int i=1;i<N;i+=2)cout<<x[i];
```

Arrays manipulation

Example: suppose that you have the following array

1	2	3	4	5	6	7	8	9	10

Write a c++ program in order to revert this array to be

10	9	8	7	6	5	4	3	2	1

Arrays manipulation

```
#include<iostream.h>
void main(){
int x[]={1,2,3,4,5,6,7,8,9,10};
int temp;// for temporary storage
for(int i=0;i<5;i++){
temp=x[i];
x[i]=x[9-i];
x[9-i]=temp;
}//for
}
```

Arrays manipulation

Example: suppose that you have the following array

1	2	3	4	5	6	7	8	9	10

Write a c++ program in order to rotate this array by one element

2	3	4	5	6	7	8	9	10	1

Arrays manipulation

```
#include<iostream.h>
void main(){
int x[10]={1,2,3,4,5,6,7,8,9,10};
temp=x[0];
for(int i=0;i<9;i++)x[i]=x[i+1];
X[9]=temp;
}
```

Search

Write a c++ program in order to search within an array for an input integer number.

- If the array is not sorted we use linear search.
- If the array is sorted we can use linear or binary search.

Linear search

```
#include<iostream.h>
void main(){
int x[10]={10,11,9,8,12,13,7,6,14,20};
int n;
cin>>n;
for(int i=0;i<10;i++){if(x[i]==n)cout<<i;break;}
}
```

Binary Search

```
#include<iostream.h>
void main(){
int x[]={1,2,3,4,5,6,7,8,9,10};
int n;cin>>n;
int start=0,end=9,mid;
mid=(start+end)/2;
while(x[mid]!=n){
if(n>x[mid])start=mid;
else if(n<x[mid])end=mid;
mid=(start+end)/2;
}//while
cout<<n<<“is located in location number”<<mid<<endl;
}
```

Maximum number of loop repetitions is logN

Passing arrays to functions

What is the output of the following program?

```
#include<iostream.h>
void t(int a[],int b[3]){
a[0]=b[0];a[1]=b[1];a[2]=b[2];
}
void main(){
int x[]={1,2,3,4}, y={5,6,7};
t(x,y);
for(int i=0;i<4;i++)cout<<x[i]<<endl;
}
```

First argument is array of any size, but the second argument is of size 3 only.

Output is:-

```
5
6
7
4
```

Because arrays are passed to functions by reference.

Array sorting

Write a c++ function in order to sort an array of any given size.

```
void sort(int a[],int size){
int temp;
for(int i=0;i<size-1;i++){
    for(int j=i+1;j<size;j++){
    if(a[j]<a[i]){
temp = a[j];
a[j]=a[i];
a[i]=temp;
}//if
}//inner
} //outer
}
```

Swap elements if a[j] < a[i] to sort the array in ascending order because j is larger than I

Swap elements if a[j]>a[i] to sort the array in descending order

Example

Correct the following code:-

```
int x=5;
int a[x];
```

The corrected code will be

```
int const x=5;
int a[x];
```

2-d arrays

- 2-d array declaration:

  type name[# of rows][# of columns]

Example:

int x[2][2];//to declare 2X2 integer array

- 2-d array filling at declaration time

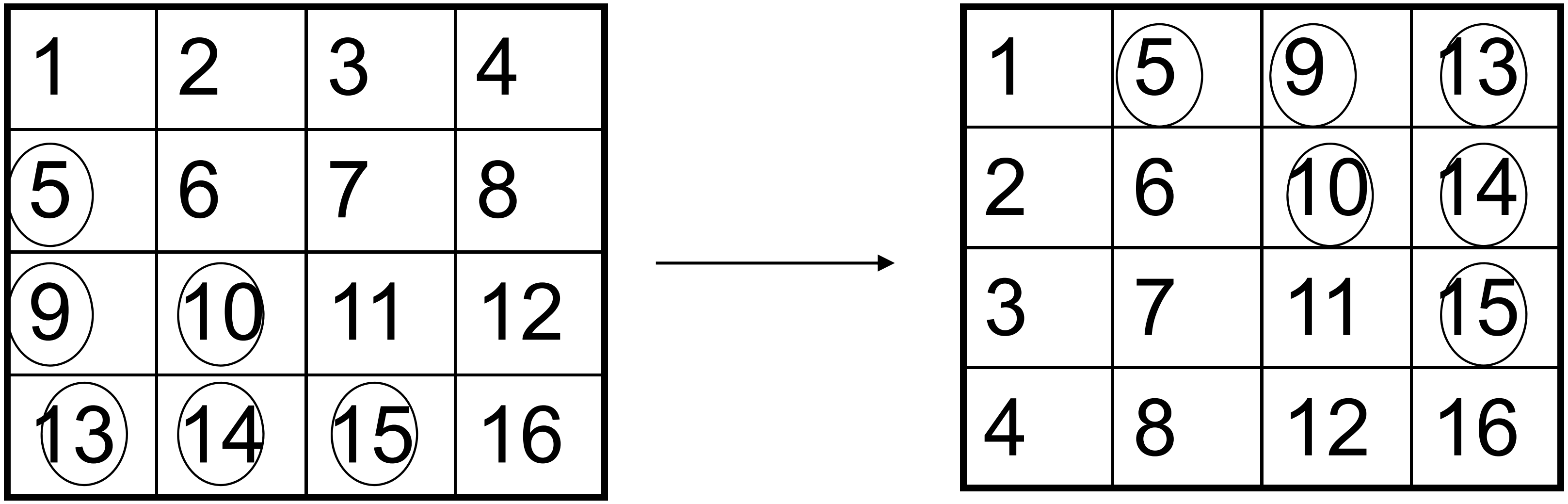
int x[3][3]={{1,2,3},{4,5,6},{7,8}};//will generate

1	2	3
4	5	6
7	8	0

If we want to manipulate a 2-d array, we need 2 nested loops, one for the rows and the other for columns.

Example

- Write a c++ program to find the transpose of a given 2-d square 4x4 integer array



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

→

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

The contents of locations below the main diagonal are exchanged with the contents of location above the main diagonal

Example

```
#include<iostream.h>
void main(){
int x[4][4]= {{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};
int temp;
for(int i=1;i<4;i++){//outer
for(int j=0;j<i;j++){//inner
temp=x[i][j];
x[i][j]=x[j][i];
x[i][j]=temp;
}
}
}
```

Matrix multiplication

Write a c++ code to multiply NXM matrix by MXK matrix.

- First matrix is considered as array a[N][M].
- Second matrix is considered as array b[M][K].
- The result will be in a third array c[N][K].

Matrix multiplication

```
int sum=0;
for(int row=0;row<N;row++){
for(int col=0;col<K;col++){
for(int elem=0;elem<M;elem++){//operation
sum+=a[row][elem]*b[elem][col];
}//operation
c[row][col]=sum;
}
}
```

Each is multiplied by each column

Character arrays

Character arrays declaration

- char x[5]={'a','h','m','a','d'};

  cout<<x;//will print ahmad╠╠╠ê

  Because there is no null character (‘\0’) at the end, but
- char x[6]={'a','h','m','a','d‘,’\0’};

  cout<<x;//will print ahmad

Character arrays

- char x[5]="ahmad";

  will generate compilation error, because this way needs a location to put a null character automatically;

- char x[6]="ahmad";

  cout<<x;// will print ahmad

  because there is a null character automatically filled.

Character arrays

Character arrays can be filled at declaration as

- Lists of characters

  If you don’t put the null character there will be no null character at the end.

- Strings

  The size of the array must be larger than the number of the letters of the string to automatically contain the null character.

Character arrays

Character arrays can be filled at execution by:-

- cin>>array_name directly.

null character will be assigned

- cin>>array_name[location_number];

  When you fill the character array element by element, there is no null character automatically assigned.

Character arrays can be passed directly to the cin or cout statements

String manipulation

- The library straing.h must be included.
- strcpy(s1,s2):- copy the characters of string s2 to s1, the size of s1 must be equal or larger than s2.
- strncpy(s1,s2,n):- copy the first n characters of s2 to s1.
- strcat(s1,s2):-concatinate s2 to.
- strncat(s1,s2,n):-concatinate the first n characters of s2 to s1.

String manipulation

- strtok(s1,token):-the remaing characters in s1 after this instruction are those before the characters contained in the second argument.
- strcmp(s1,s2):-compare s1 and s2 character by character until a difference is found. If the character of s1 has higher ascii value it returns 1. if that of s2 is higher, it returns -1. if there is no change, it returns 0.

String manipulation

- strncmp(s1,s2,n):- if you want to compare the first n characters only.

- strlen(s1):- determines the number of characters preceding the null character.

example

- What is the output of the following code?

```
char k1[40],k2[20]=“c++ programming”;
strcpy(k1,k2);
cout<<k1;//will print c++ programming.
strtok(k2,”p”);
cout<<k2;//c++
strcat(k1,k2);
cout<<k1;//c++ programmingc++
cout<<strlen(k1);//will print 19
```

Since there is a space after the c++ in k2 when strtok statement have been executed

2-d character arrays

Each row in the 2-d character arrays can be treated as if it is 1-d character array in execution.

For example:-

```
char x[5][20];
```

Can contain 5 strings of maximum length 20 characters for each.

Example

What is the output of the following code?

```
char x[5][20];
cin>>x[0];//if you entered “computer”
cout<<x[0];//will print computer
```

Pointers

Pointer variables:-

- Contain memory address as values.
- Pointers contain address of variable.

Pointer declaration:-

- int *mypointer;

We can read this declaration statement from right to left as:-

mypointer is a pointer to integer location.

Pointers

Pointer initialization:-

- int x=5, *xptr=&x;

Makes the pointer xptr points to location x, or store the address of x in location xptr.

- int *xptr = NULL;

Makes the pointer xptr points to nothing.

The pointer type must match the location type that points to.

Pointers

What is the output of the following code?

```
int x=5, *xptr=&x;
*xptr = 7;
cout<<*xptr<<endl;//will print 7
```

* Before the variable name at declaration means that the variable is a pointer

* Before the variable name at execution means that the statement access the content of the location that the pointer points to.

Pointers

What is the output of the following code?

int x=5, *xptr =&x;

cout<<*(&x);//will print 5

Means that print the content of location that has the address of x

In any statement the existence of * and & will cancel each other

Pointers and arrays

What is the output of the following code?

```
int x[5]={1,2,3,4,5}, *xptr=x;
xptr+=2;//move pointer forward by 2 locations
cout<<xptr[0]<<endl;//print 3
```

Because each array is a static pointer points to the first location in the array.

xptr[0] means the content of the location that the pointer points to.

Pointers and arrays

- What is the output of the following code?

```
int x[5]={1,2,3,4,5},*xptr=x;
xptr+=2;
cout<<xptr[-1]<<endl;//print 2
```

xptr[-1] means that the content of the location that behind the location the pointer xptr points to

Pointers to character arrays

- What is the output of the following code?

```
char x[10]=“ahmad”;
char *xptr=x;
cout<<x+2;// prints “mad”
cout<<*(x+2);//prints ‘m’
```

Printing the character pointers prints the contents of all locations until a null character is encountered.

But printing the content of a character pointer prints the content of the location that the pointer points to.

Functions

- Construct a program from smaller pieces or components.
- Each piece is more manageable than the original program.
- Functions are invoked by function calls.
- Functions written once and can be used many.
- Function calls can be exist inside another function.

Functions

Functions are of two types:-

- User defined functions.
- Build-in C++ functions like math library functions to perform mathematical calculations.

Will print 4

Example:-

```
cout<< sqrt(16)<<endl;
```

To perform this operation, the file math.h must be included

Math library functions

Method	Description	Example

`ceil( x )`	rounds *x* to the smallest integer not less than *x*	`ceil( 9.2 )` is `10.0`
		`ceil( -9.8 )` is `-9.0`

`cos( x )`	trigonometric cosine of *x* (*x* in radians)	`cos( 0.0 )` is `1.0`

`exp( x )`	exponential function *ex*	`exp( 1.0 )` is `2.71828`
		`exp( 2.0 )` is `7.38906`

`fabs( x )`	absolute value of *x*	`fabs( 5.1 )` is `5.1`
		`fabs( 0.0 )` is `0.0`
		`fabs( -8.76 )` is `8.76`

`floor( x )`	rounds *x* to the largest integer not greater than *x*	`floor( 9.2 )` is `9.0`
		`floor( -9.8 )` is `-10.0`

`fmod( x, y )`	remainder of *x*/*y* as a floating-point number	`fmod( 13.657, 2.333 )` is `1.992`

`log( x )`	natural logarithm of *x* (base *e*)	`log( 2.718282 )` is `1.0`
		`log( 7.389056 )` is `2.0`

`log10( x )`	logarithm of *x* (base 10)	`log10( 10.0 )` is `1.0`
		`log10( 100.0 )` is `2.0`

`pow( x, y )`	*x* raised to power *y* (*xy*)	`pow( 2, 7 )` is `128`
		`pow( 9, .5 )` is `3`

`sin( x )`	trigonometric sine of *x* (*x* in radians)	`sin( 0.0 )` is `0`

`sqrt( x )`	square root of *x*	`sqrt( 900.0 )` is `30.0`
		`sqrt( 9.0 )` is `3.0`

`tan( x )`	trigonometric tangent of *x* (*x* in radians)	`tan( 0.0 )` is `0`

Fig. 3.2 Math library functions.

Local variables vs parameters

- Local variables: known only in the function in which they are defined.

- Parameters:- local variables passed to function when called in order to provide outside information.

Function definition

- Function Prototype:- tells compiler the arguments type and the type of the function

Type of the function

Arguments type

Example:- int add(int,int);

- Function Body:-

Statements that construct the function

- Function Call:-

The statement that invoke the function and pass arguments to the function.

Example

Using functions, write a C++ program in order to add two integer numbers.

```
#include<iostream.h>
int add(int, int);
void main(){
int x=5,y=7;
cout<<add(x,y)<<endl;
}
int add(int a, int b){
return a+b;
}
```

Function prototype

Function call

Function body

Function prototype

- From the previous example, the syntax of function prototype is:

return_type name(arg1_type,arg2_type,…..);

- It may contain dummy variable names, for example:- ***int add(int w,int z);***
- The absence of function prototype generates compilation error. Because the function prototype tells the compiler about the function return type, function name, and about the arguments type.
- If the function does not return results, its return type is *void*

Function body

- The syntax of the function body is

return_type name(arg1_type arg1,…argn_type argn){
statemets
}

These arguments are to receive the values passed from the function call

- The absence of function body generates linking error.
- Return type is void if the function does not return a result.

Function call

- Functions called by writing

functionName(arg1,arg2,…,argn);

Example: cout<<add(x,y)<<endl;

Or cout<<add(2,3)<<endl;

- The absence of function call generates runtime error, such that the function will not be executed.

Function signature

Different functions considered as different if they are different in:-

- Name

Or

- Arguments type

Or

- Number of arguments

Or

- The arguments order.

Default arguments

- What about if you want a function to add three numbers, and another to add four numbers and so on?
- So you need to write different functions, such that, one for each number of arguments.
- What about to write one function that can take 4 or 3 or 2 arguments

Example

```
#include<iostream.h>
int add(int=0,int=0,int=0,int=0);
void main(){
cout<<add(2,3)<<endl;
cout<<add(2,3,4)<<endl;
cout<<add(2,3,4,5)<<endl;
}
int add(int a, int b, int c, int d){
return a+b+c+d; }
```

If there is no value passed for the argument, then the specified default value is assigned

Function overloading

- Can two functions or more have the same name?

- The answer is yes, if they are different in function signature.

Example

```
#include<iostream.h>
void add(int,double);
void add(double,int);
void main(){
add(2,3.5);
add(2.5,3);
}
void add(int a,double b){cout<<“fun1\n”;}
void add(double a,int b){cout<<“fun2\n”;}
```

The output is:-

```
fun1
fun2
```

You can not write add(2,3) because it makes a conflict

Inline functions

```
#include<iostream.h>
int add(int a,int b){return a+b;}
void main(){
cout<<add(2,3)<<endl;
}
```

Inline function, the body and the prototype are written together

Inline functions are faster in execution

Recursion

Recursive functions:-

- Functions can call themselves.
- Slowly converges toward a base case.
- Function makes call to itself inside the return statement.
- When the base case get solved, answer works way back to solve the entire problem.

Example…factorial

```
#include<iostream.h>
int fact(int a){
if(a==1)return 1;
else return a*fact(a-1); }
void main(){
cout<<fact(5)<<endl;
}
```

The base case is 1 because the return value is known. Then the answer goes back to solve the entire problem.

Random number generation

- The function rand() is used to generate random numbers.

- The file stdlib.h must be included.

- It can be used to generate random numbers within a certain range for certain application, example :1..6 in playing dice

Random number generation

```
#include<iostream.h>
#include<stdlib.h>
void main(){
for(int i=0;i<=10;i++){
cout<<rand()<<endl;}
}
```

When you run this program you will see 10 random numbers.

But the surprise is, every time you run this program you will see the same sequence. So are they random?

Random number generation

```
#include<iostream.h>
#include<stdlib.h>
void main(){
int seed;
cin>>seed;
srand(seed);
for(int i=0;i<=10;i++){
cout<<rand()<<endl;}
}
```

When you run the program with different seeds, you will get different sequences

Random number generation

If you want the random numbers to be within a specific range, you need to use the following equation:-

x+rand()%y

Where x: is the lower number in the range.

y: is the number of items in the range

Example: -10+rand()%20 will generate random numbers in the range -10..9

Aliasing

- Reference variables are declared as follows:- int x=5; int &y=x;
- From the &y=x, you can alter x by two ways:-

x++ will increment x by one

y++ also will increment x by one.

- Reference variables must be initialized.

Example

What is the output of the following code?

```
int x=7, &y=x;
cout<<y<<endl;
x++;
cout <<y<<endl;
y++;
cout<<x<<endl;
```

```
7
8
9
```

Call by reference and call by value

Call by value

- Copy of data passed to function.
- Changes to copy do not change original.

Call by reference

- Function can directly access data.
- Changes affect original.

Example

```
#include<iostream.h>
int square(int &a){
a++;cout<<“square=”;return a*a;}
void main(){
int x=2;
cout<<square(x)<<endl;
cout<<“x=”<<x<<endl;
}
```

Output is:-

Square=9

x=3

Because argument (a) becomes an alias for variable (x). so the changes done for argument a inside the function affect the variable x.

You can notice that the statements inside the function finish execution before the calling statement.

Static variables

- They are automatically initialized to zero.
- If they exist in a function, they keeps value between function calls.

Example:- what is the output of the following program?

Example

```
#include<iostream.h>
void test();
void main(){
test();
test();
test();
}
void test(){
int static st_x=2;int nst_x=2;
st_x++;nst_x++;
cout<<“non static=”<<nst_x<<“static=”<<st_x<<endl;
}
```

The output is:-

```
non static=3        static=3
non static=3        static=4
non static=3        static=5
```

Local and global variables

- Local variables are declared within a specific scope and can be accessed only from inside that scope, like inside a function.
- Global variables are declared outside the functions, or within a scope such that multiple smaller scopes can access the variables.

Usually, if we have shared value, we put it inside a global variable.

Example

What is the output of the following program?

```
#include<iostream.h>
int x=5;
void t1(){x++;}
void t2(){x++;}
void main(){
t1();
t2();
x++;
cout<<x<<endl;}
```

Each function can alter the same variable.

So the output is 8